The Optimal Route: A Rigorous Survey of Foundational Shortest-Path Algorithms

Tai Vu
Stanford University
taivu@stanford.edu

1 Introduction

From the GPS in our cars to the vast web of global internet traffic, the ability to find the most efficient route between two points is a cornerstone of modern technology and a fundamental problem in computer science. At its core, this is a question of finding the shortest path in a network, or graph. While seemingly simple, the challenge is immense. A brute-force approach, which would involve calculating the length of every possible path between two nodes, becomes computationally intractable with even a modest increase in the size of the network, as the number of paths can grow exponentially. This computational barrier necessitates the development of sophisticated algorithms that can intelligently navigate a graph to find the optimal path without an exhaustive search.

This paper provides a rigorous survey and analysis of three canonical algorithms that form the foundation of shortest-path computation: **Dijkstra's Algorithm**, the **Bellman-Ford Algorithm**, and the **Floyd-Warshall Algorithm**. Each of these methods offers a unique solution tailored to specific constraints and problem variations. Dijkstra's algorithm provides an efficient solution for graphs with non-negative edge weights, a common scenario in applications like route planning. The Bellman-Ford algorithm extends this capability to handle graphs with negative edge weights, which are crucial for modeling more complex systems like financial transactions. Finally, the Floyd-Warshall algorithm addresses the all-pairs shortest path problem, efficiently computing the optimal route between every pair of vertices in a graph.

For each of these seminal algorithms, this paper will provide a detailed exposition of its mechanics, a formal proof of its correctness, and a thorough analysis of its asymptotic time complexity. Through this comprehensive examination, we aim to illuminate the elegance and power of these foundational tools in graph theory.

2 Problem Description

Before delving into the problem of finding shortest paths, we need concrete definitions of edge lengths and path lengths in a graph. We will only consider directed graph in this paper, because undirected graph can be regarded as a special case of directed graph with two bidirectional edges between each pair of vertices.

Definition 2.1. Let $G = (V, E)$ be a directed graph. G is called a weighted graph if there is a function $\ell : E \to \mathbb{R}$ that map each edge to a real-valued weight. ℓ is called a weight function of G. For each edge $(u, v) \in E$, $\ell(u, v)$ is called the weight (or length) of (u, v).

Definition 2.2. Let $G = (V, E)$ be a directed graph. A path in G is a sequence of vertices $v_0, v_1, \ldots, v_k$ and a sequence of edges $(v_i, v_{i+1}) \in E$ for $0 \leq i \leq k-1$. We denote such a path by $p = (v_0, v_1, \ldots, v_k)$, and p is called a $v_0 \to v_k$ path. If for such a path with $k \geq 1$, v_k, v_0 is also an edge in G, then $v_0, v_1, \ldots, v_k, v_0$ is a cycle.

Definition 2.3. Given a directed graph $G = (V, E)$, for any pair of vertices u, v, let $P(u, v)$ denote the set of all $u \to v$ paths in G.

The above definitions allow for repeated vertices and edges along a path, as this is common in many computer science applications. Next, we have a definition of path lengths and shortest paths in a graph.

Definition 2.4. Let $G = (V, E)$ be a weighted, directed graph with weight function $\ell : E \to \mathbb{R}$. The weight (or length) $\ell(p)$ of a path $p = (v_0, v_1, \ldots, v_k)$ is the sum of the weights of all the edges in p.

$$\ell(p) = \sum_{i=0}^{k-1} \ell(v_i, v_{i+1})$$

Definition 2.5. Let $G = (V, E)$ be a weighted, directed graph with weight function $\ell : E \to \mathbb{R}$. For any pair of vertices u, v in G, the shortest-path weight (or shortest-path length) $\delta(u, v)$ from u to v is defined as

$$\delta(u, v) = \begin{cases} \min\limits_{p \in P(u,v)} \ell(p) & \text{if there exists a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

A shortest path from u to v is any $u \to v$ path p such that $\ell(p) = \delta(u, v)$.

In the upcoming sections of the paper, we aim to solve the question of searching for shortest paths, which means paths of minimum weights between two vertices. More formally, we define the Single-Source Shortest Path problem as follows.

Problem 2.6 (Single-Source Shortest Paths)**.** Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$ and $|E| = m$. Given a source vertex $s \in V$, find a shortest path (and its weight) from s to every vertex $v \in V$ (Cormen et al., 2009).

In an unweighted graph (a special case in which every edge has a weight of 1), the Single-Source Shortest Path problem can be solved using Breadth-First Search (Cormen et al., 2009). In particular, Breadth-First Search simply explores every k-hop neighborhood of the source vertex in an increasing order of k, until every vertex is visited. This approach has a quick asymptotic time complexity of $O(m + n)$. However, many real-world applications like Google Map involves graph representations that contain edges of different weights, which Breadth-First Search fails to solve. Therefore, the next two sections will present two efficient algorithms for dealing with this problem, including the Dijkstra's Algorithm and the Bellman-Ford Algorithm.

Furthermore, we will examine a more general version of the shortest path finding challenge, which is the All-Pairs Shortest Path problem. Section 5 will discuss the Floyd-Warshall Algorithm for answering this question.

Problem 2.7 (All-Pairs Shortest Paths)**.** Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$ and $|E| = m$. Find a shortest path (and its weight) from u to v for every pair of vertices $u, v \in V$ (Cormen et al., 2009).

3 Dijkstra's Algorithm

Dijkstra's Algorithm is a popular approach for solving the Single-Source Shortest Path problem (Problem 2.6). This method works when the input graph has non-negative edge weights. Such an assumption is realistic in a number of real-world applications, such as route planning systems in Google Map and Uber.

For convenience, we will also assume that every vertex is reachable from the source vertex s. In other words, there is a path from s to v for any $v \in V$. In fact, given a general input graph, we can simply run a fast graph traversal technique like Depth-First Search or Breadth-First Search and then remove all the vertices that are not reachable from s.

Now, we can state the Dijkstra's Algorithm.

Algorithm 3.1 (Dijkstra's Algorithm)**.** Let $G = (V, E)$ be a weighted, directed graph with weight function $\ell : E \to \mathbb{R}$ and a source vertex s. Suppose that G has non-negative edge weights. Let X denote the set of vertices that have been processed. Let A and B be two arrays such that A_u will represent the weight of the shortest $s \to u$ path, while B_u will represent the shortest $s \to u$ path for every $u \in V$. The Dijkstra's Algorithm proceeds as follows.

- Initialize $X = \{s\}$.
- Initialize $A_s = 0$ and $A_u = \infty$ for all $u \in V \setminus \{s\}$.
- Initialize $B_s = (s)$ and $B_u = ()$ for all $u \in V \setminus \{s\}$ (i.e. an empty path).
- While there exists an edge $(v, w) \in E$ such that $v \in X, w \in V \setminus X$, do the following steps:
  - Compute $A_v + \ell(v, w)$ for each edge $(v, w) \in E$ where $v \in X, w \in V \setminus X$.
  - Pick (v^*, w^*) such that $A_{v^*} + \ell(v^*, w^*) = \min_{(v,w)\in E, v\in X, w\in V\setminus X} \{A_v + \ell(v, w)\}$.
  - Add w^* to X.
  - Update $A_{w^*} := A_{v^*} + \ell(v^*, w^*)$.
  - Update $B_{w^*} := B_{v^*} \cup (v^*, w^*)$.
- Return A and B.

(Roughgarden, 2018; Cormen et al., 2009)

The following theorem demonstrates that the Dijkstra's Algorithm produces desired outputs when all the edge weights of the input graph are non-negative.

Theorem 3.2 (Correctness of Dijkstra's Algorithm)**.** *Let $G = (V, E)$ be a weighted, directed graph with weight function $\ell : E \to \mathbb{R}$ and a source vertex s. Suppose that G has non-negative edge weights. Then, at the conclusion of the Dijkstra's Algorithm, we have $A_u = \delta(s, u)$ and B_u is a shortest $s \to u$ path for all $u \in V$ (Roughgarden, 2018).*

Proof. Let $P(t)$ denote the statement that $A_u = \delta(s, u)$ and B_u is a shortest $s \to u$ path for all $u \in X$ after t iterations. We will prove, by induction, that $P(t)$ is true for all $0 \leq t \leq n - 1$.

For the base case, we have $X = \{s\}$ at the beginning of the algorithm, while $A_s = 0 = \delta(s, s)$ and $B_s = (s)$. Thus, $P(0)$ is true. For the inductive step, assume that $P(t - 1)$ is true for some $1 \leq t \leq n - 1$. We need to show that $P(t)$ is true.

According to the Dijkstra's Algorithm, we need to pick an edge (v^*, w^*) such that $A_{v^*} + \ell(v^*, w^*) = \min_{(v,w)\in E, v\in X, w\in V\setminus X} \{A_v + \ell(v, w)\}$. Then, we add w^* to X and update $A_{w^*} = A_{v^*} + \ell(v^*, w^*)$ and $B_{w^*} = B_{v^*} \cup (v^*, w^*)$. Thus, we only need to show that $A_{w^*} = \delta(s, w^*)$ and B_{w^*} is a shortest $s \to w^*$ path.

From the inductive hypothesis, we know that $A_{v^*} = \delta(s, v^*)$ and B_{v^*} is a shortest $s \to v^*$ path. This means $\ell(B_{v^*}) = \delta(s, v^*) = A_{v^*}$. So we have

$$\ell(B_{w^*}) = \ell(B_{v^*}) + \ell(v^*, w^*) = A_{v^*} + \ell(v^*, w^*) = A_{w^*}$$

Since B_{w^*} forms a path from s to w^*, we must have $A_{w^*} = \ell(B_{w^*}) \geq \delta(s, w^*)$ (according to the definition of δ).

Next, we will show that $\delta(s, w^*) \geq A_{w^*}$. To do so, let P be a shortest $s \to w^*$ path, which means $\ell(P) = \delta(s, w^*)$.

P starts at $s \in X$ and ends at $w^* \in V \setminus X$. Thus, we choose x to be the last vertex of P such that $x \in X$. Let y denote the subsequent vertex following x in P, then $y \in V \setminus X$. Let P_1, P_2, P_3 denote the $s \to x$, $x \to y$, $y \to w^*$ components of P. This tells us that $\ell(P) = \ell(P_1) + \ell(P_2) + \ell(P_3)$.

Because all the edges in G have non-negative weights, we have $\ell(P_3) \geq 0$. Additionally, since P_2 is the edge (x, y), we know that $\ell(P_2) = \ell(x, y)$. Meanwhile, since P_1 is a path from s to x, we have $\ell(P_1) \geq \delta(s, x)$. The inductive hypothesis implies $A_x = \delta(s, x)$, so we learn that $\ell(P_1) \geq A_x$. This tells us that

$$\ell(P) = \ell(P_1) + \ell(P_2) + \ell(P_3) \geq A_x + \ell(x, y) + 0 = A_x + \ell(x, y)$$

According to the way we select (v^*, w^*), we know that $A_{v^*} + \ell(v^*, w^*) \leq A_x + \ell(x, y)$. Hence, we can see that $\ell(P) \geq A_{v^*} + \ell(v^*, w^*) = A_{w^*}$, which means $\delta(s, w^*) \geq A_{w^*}$.

Therefore, we learn that $\delta(s, w^*) = A_{w^*} = \ell(B_{w^*})$. In other words, we can see that $A_{w^*} = \delta(s, w^*)$ and B_{w^*} is a shortest $s \to w^*$ path. After adding w^* to X, we learn that $A_u = \delta(s, u)$ and B_u is a shortest $s \to u$ path for all $u \in X$ after t iterations. Thus, $P(t)$ is true, completing the induction.

We start with $|X| = 1$ and add 1 vertex to X in each iteration until $X = V$, so there are $n - 1$ iterations. Since $P(n - 1)$ is true, we can conclude that at the conclusion of the Dijkstra's Algorithm, we have $A_u = \delta(s, u)$ and B_u is a shortest $s \to u$ path for all $u \in V$. □

Subsequently, we will examine the time complexity of the Dijkstra's Algorithm.

Theorem 3.3 (Straightforward Running Time of Dijkstra's Algorithm)**.** *Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$ and $|E| = m$, and G has non-negative edge weights. Then, a straightforward implementation of the Dijkstra's Algorithm runs in $O(mn)$ time (Roughgarden, 2018).*

Proof. Notice that in practice, we actually do not need to store and update the whole array B. In fact, we only need to store a predecessor of v on a shortest path from s, and then update this value in each iteration. At the conclusion of the algorithm, these predecessors can be used to easily construct the entire shortest paths. In this way, the update time for each entry of B becomes $O(1)$.

We start with $|X| = 1$ and add 1 vertex to X in each iteration, so there are $n - 1$ iterations. In each iteration, a straightforward implementation conducts an an exhaustive search over all the crossing edges from X to $V \setminus X$. It computes a score $A_v + \ell(v, w)$ for each crossing edge (v, w) and then picks the one with minimum score. There are $O(m)$ such crossing edges, so each iteration requires $O(m)$ work. Therefore, the straightforward implementation runs in $O(mn)$ time. □

A running time of $O(mn)$ time is pretty decent. However, for dense graphs, this time complexity becomes approximately $O(n^3)$, which can be slow when the number of vertices is extremely large. The question is: Can we do better than $O(mn)$? The key idea is speeding up the process of computing and extracting a crossing edge with a minimum score using an appropriate data structure. The following theorem illustrates this idea.

Theorem 3.4 (Improved Running Time of Dijkstra's Algorithm)**.** *Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$ and $|E| = m$, and G has non-negative edge weights. Then, there is an implementation of the Dijkstra's Algorithm that runs in $O((m + n) \log n)$ time (Roughgarden, 2018).*

Proof. We will use a heap data structure to store the vertices of $V \setminus X$. The reason is that the heap data structure supports fast insertion and extraction of the smallest element. Concretely, our heap must maintain the following invariants throughout the algorithm.

- The elements in the heap are all the vertex in $V \setminus X$.
- For each $u \in V \setminus X$, the heap stores u along with its value

$$s(u) = \min_{(v,u) \in E, v \in X} \{A_v + \ell(v, u)\}$$

To maintain the second invariant, we proceed as follows. After selecting w^* and removing it from the heap, we consider every edge (w^*, u) such that $u \in V \setminus X$. We will delete u from the heap, recompute $s(u) := \min\{s(u), A_{w^*} + \ell(w^*, u)\}$, and then insert it back to the heap. The detailed implementation and pseudocode can be found in Roughgarden (2018).

Now, most of the work is dominated by heap-based operations. Inserting an element to the heap takes $O(\log n)$ time, and removing the element with the smallest value requires $O(\log n)$ time. We need $n - 1$ removal operations to remove all the vertices in $V \setminus X$. In addition, each edge in the input graph creates at most a pair of removal and insert operations. Thus, the number of heap operations is $O(m + n)$, so the implementation runs in $O((m + n) \log n)$ time. □

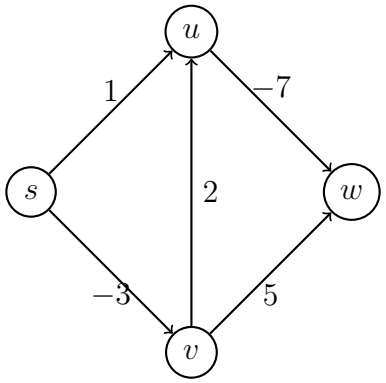
Remark 3.5. *If the input graph G is weakly connected, then the running time becomes $O((m+n)\log n) = O(m \log n)$, because $m \geq n-1$.*

Remark 3.6. *We can achieve a better amortized running time of $O(n \log n + m)$ by using a Fibonaaci heap (Cormen et al., 2009).*

4 Bellman-Ford Algorithm

The proof of correctness for the Dijkstra's Algorithm requires the condition that the input graph has no negative edge weights. This condition is necessary, because Dijkstra's Algorithm fails when there are edges with negative weights, which can be illustrated in the following example.

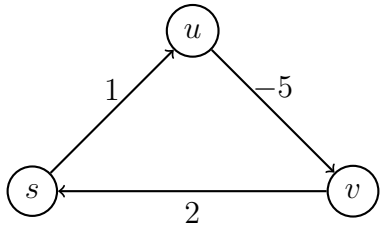
Example 4.1. Consider the following graph.



Dijkstra's Algorithm returns the following $s \to w$ path: (s, u, w). This has a weight of -6. However, the true shortest $s \to w$ path is (s, v, u, w), which has a weight of -8.

Even though most real-life graph models consist of non-negative edge weights, there are some applications that require handling negative edges, such as tracking financial transactions. Thus, we need another approach to tackle the graph with negative edge weights. However, when negative edge weights appear in a graph, defining shortest paths might become a challenge, as shown below.

Example 4.2. Consider the following graph.



What is the shortest $s \to u$ path? If we follow the edge (s, u) and then go around the cycle k times, we obtain a $s \to u$ path of weight $1 + k(1 - 5 + 2) = 1 - 2k$, which approaches $-\infty$ as $k \to \infty$. Thus, the shortest $s \to u$ path is undefined.

As we can see from Example 4.2, with the presence of negative cycles (i.e. cycles of negative weights), the concept of shortest paths cannot be well defined. Therefore, to solve the problem of finding shortest paths, we need to assume that the input graph contains no negative cycles.

Now, we have a proposition that will narrow down the search space for shortest paths to the set of paths with no more than $n - 1$ edges.

Proposition 4.3. *Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$. Suppose that there are no negative cycles in G. Then, if there is a path from u to v, then there is a shortest $u \to v$ path with at most $n - 1$ edges (Roughgarden, 2019).*

Proof. Let P denote a shortest $u \to v$ path with the smallest number of edges. We have $P = (v_0, v_1, \ldots, v_k)$ where $v_0 = u$ and $v_k = v$.

Assume for the sake of contradictions that P contains at least n edges. This means P has at least $n + 1$ vertices, so there must be repeated vertices along P, which means $v_i = v_j$ for some $0 \leq i < j \leq k$.

Since G has no negative cycles, the cycle from v_i to v_j must have non-negative weight. By removing that cycle, we obtain a new $u \to v$ path $P' = (v_0, v_1, \ldots, v_{i-1}, v_i = v_j, v_{j+1}, \ldots, v_k)$ that has $\ell(P') \leq \ell(P)$. However, since P is a shortest $u \to v$ path, we know that $\ell(P) \leq \ell(P')$. This tells us that $\ell(P') = \ell(P)$, so P' is a shortest $u \to v$ path with fewer edges than P, which is a contradiction.

We have reach a contradiction, so our assumption must have been wrong. Hence, P contains at most $n - 1$ edges. Therefore, if there is a path from u to v, then there is a shortest $u \to v$ path with at most $n - 1$ edges. □

Bellman-Ford Algorithm is a well-known solution to the Single-Source Shortest Path problem when negative edge weights are present. This approach leverages the idea of dynamic programming. Specifically, to solve a given problem, we will break it down to subproblems with smaller size and same structure (which is called *optimal substructure*). By solving the subproblems recursively, we obtain a solution for the main problem. The Bellman-Ford Algorithm is stated as below.

Algorithm 4.4 (Bellman-Ford Algorithm)**.** Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$, weight function $\ell : E \to \mathbb{R}$, and a source vertex s. Let A be a $n \times n$ array such that $A_{k,v}$ will represent the minimum length of an $s \to v$ path with at most k edges for each $0 \leq k \leq n - 1$ and $v \in V$. In addition, let B be a $n \times n$

array such that $B_{k,v}$ will represent a shortest $s \to v$ path with at most k edges for each $0 \leq k \leq n-1$ and $v \in V$. The Bellman-Ford Algorithm proceeds as follows.

- Initialize $A_{0,s} = 0$ and $A_{0,v} = \infty$ for all $v \in V \setminus \{s\}$.
- Initialize $B_{0,s} = (s)$ and $B_{0,v} = ()$ for all $v \in V \setminus \{s\}$.
- For $1 \leq k \leq n-1$ and $v \in V$, do the following steps:
  - Run the updates

$$A_{k,v} = \min \left\{ \begin{array}{c} A_{k-1,v} \\ \min\limits_{(w,v) \in E} \{A_{k-1,w} + \ell(w,v)\} \end{array} \right\}$$

  - If $A_{k,v} = A_{k-1,v}$, then set $B_{k,v} = B_{k-1,v}$. Otherwise, we find a vertex $w' \in V$ such that $(w', v) \in E$ and $A_{k,v} = A_{k-1,w'} + \ell(w', v)$, then set $B_{k,v} = B_{k-1,w'} \cup (w', v)$.
- Return $A_{n-1,v}$ and $B_{n-1,v}$ for all $v \in V$.

(Roughgarden, 2019; Cormen et al., 2009)

To prove the correctness of the Bellman-Ford Algorithm, we need to examine its optimal substructure.

Lemma 4.5 (Bellman-Ford Optimal Substructure)**.** *Let $G = (V, E)$ be a weighted, directed graph with a source vertex s. For some $k \geq 1$ and $v \in V$, let P denote a shortest $s \to v$ path in G with at most k edges. Then, one of the following cases must occur.*

- *P is a shortest $s \to v$ path in G with at most $k-1$ edges.*
- *For some $w \in V$, P is obtained by appending the edge (w, v) to a shortest $s \to w$ path in G with at most $k-1$ edges.*

(Roughgarden, 2019)

Proof. There are two cases.

- Case 1: P has at most $k-1$ edges.

  Let Q be a shortest $s \to v$ path in G with at most $k-1$ edges. Since Q has no more than k edges, we have $\ell(Q) \geq \ell(P)$ by applying the definition of P. However, because P has no more than $k-1$ edges, we have $\ell(P) \geq \ell(Q)$ by applying the definition of Q. Thus, we learn that $\ell(P) = \ell(Q)$, so P is a shortest $s \to v$ path in G with at most $k-1$ edges.

- Case 2: P has exactly k edges.

  Let w be the vertex right before v in P. Let P' denote the $s \to w$ component of P. Then, P' is a $s \to w$ path with $k-1$ edges, and $\ell(P) = \ell(P') + \ell(w, v)$.

  In this case, we will show that P' is a shortest $s \to w$ path with at most $k-1$ edges. In fact, assume that there is a shorter $s \to w$ path with at most $k-1$ edges, denoted by Q'. Then, the combined path $Q = Q' \cup (w, v)$ has weight $\ell(Q) = \ell(Q') + \ell(w, v) < \ell(P') + \ell(w, v) = \ell(P)$, so Q is a $s \to v$ path with at most k edges that is shorter then P, which is a contradiction. Hence, P' is a shortest $s \to w$ path with at most $k-1$ edges.

Therefore, P is either a shortest $s \to v$ path in G with at most $k-1$ edges, or obtained by appending the edge (w, v) to a shortest $s \to w$ path in G with at most $k-1$ edges for some $w \in V$. □

This lemma gives us a convenient recurrence relation to compute shortest path weights.

Corollary 4.5.1 (Bellman-Ford Recurrence)**.** *Let $G = (V, E)$ be a weighted, directed graph with a source vertex s. For each $k \geq 0$ and $v \in V$, let $L_{k,v}$ denote the minimum length of an $s \to v$ path with at most k edges ($L_{k,v} = \infty$ if there are no such paths). Then, for every $k \geq 1$ and $v \in V$,*

$$L_{k,v} = \min \left\{ \begin{array}{c} L_{k-1,v} \\ \min\limits_{(w,v) \in E} \{L_{k-1,w} + \ell(w, v)\} \end{array} \right\}$$

(Roughgarden, 2019)

Proof. Let P denote a shortest $s \to v$ path in G with at most k edges. This means $\ell(P) = L_{k,v}$. Lemma 4.5 implies the following two cases.

- Case 1: P is a shortest $s \to v$ path in G with at most $k-1$ edges.

  In this case, we have $\ell(P) = L_{k-1,v}$, so we get $L_{k,v} = L_{k-1,v}$

- Case 2: For some $w^* \in V$, P is obtained by appending the edge (w^*, v) to a shortest $s \to w^*$ path in G with at most $k-1$ edges.

  In this case, we have $\ell(P) = L_{k-1,w^*} + \ell(w^*, v)$, so we get $L_{k,v} = L_{k-1,w^*} + \ell(w^*, v)$.

  Assume that $L_{k-1,w^*} + \ell(w^*, v) \neq \min\limits_{(w,v) \in E} \{L_{k-1,w} + \ell(w, v)\}$. In other words, there exists $(w', v) \in E$ such that $L_{k-1,w^*} + \ell(w^*, v) > L_{k-1,w'} + \ell(w', v)$.

  Let Q' denote the shortest $s \to w'$ path with at most $k-1$ edges, then $Q = Q' \cup (w', v)$ is a $s \to v$ path with at most k edges. So the weight of Q is $\ell(Q) = L_{k-1,w'} + \ell(w', v) < L_{k-1,w^*} + \ell(w^*, v) = \ell(P)$, which contradicts the

minimality of P. Hence, we have $L_{k-1,w^*} + \ell(w^*, v) = \min_{(w,v)\in E}\{L_{k-1,w} + \ell(w, v)\}$, which means $L_{k,v} = \min_{(w,v)\in E}\{L_{k-1,w} + \ell(w, v)\}$.

Therefore, we can conclude that for every $k \geq 1$ and $v \in V$,

$$L_{k,v} = \min \left\{ \begin{array}{c} L_{k-1,v} \\ \min_{(w,v)\in E}\{L_{k-1,w} + \ell(w, v)\} \end{array} \right\}$$

□

Finally, we will rigorously prove the correctness of the Bellman-Ford Algorithm in the next theorem.

Theorem 4.6 (Correctness of Bellman-Ford Algorithm)**.** *Let $G = (V, E)$ be a weighted, directed graph with weight function $\ell : E \to \mathbb{R}$ and a source vertex s. Suppose that there are no negative cycles in G. Then, at the conclusion of the Bellman-Ford Algorithm, we have $A_{n-1,v} = \delta(s, v)$ and $B_{n-1,v}$ is a shortest $s \to v$ path for all $v \in V$ (Roughgarden, 2019).*

Proof. First, for each $0 \leq k \leq n-1$ and $v \in V$, let $L_{k,v}$ denote the minimum length of an $s \to v$ path with at most k edges. By Corollary 4.5.1, we learn that

$$L_{k,v} = \min \left\{ \begin{array}{c} L_{k-1,v} \\ \min_{(w,v)\in E}\{L_{k-1,w} + \ell(w, v)\} \end{array} \right\}$$

We can show that $L_{n-1,v} = \delta(s, v)$ for all $v \in V$. In fact, if v is not reachable from s, then $L_{n-1,v} = \delta(s, v) = \infty$. Otherwise, since there are no negative cycles in G, we can find a shortest $s \to v$ path with at most $n-1$ edges (by Proposition 4.3). Let P denote such a path.

By the definition of $L_{n-1,v}$, we have $L_{n-1,v} \leq \ell(P)$. However, because P is a shortest $s \to v$ path, we know that $L_{n-1,v} \geq \ell(P)$. Thus, we learn that $L_{n-1,v} = \ell(P) = \delta(s, v)$. Consequently, we can see that $L_{n-1,v} = \delta(s, v)$ for all $v \in V$.

Next, let $P(k)$ denote the statement that $A_{k,v} = L_{k,v}$ for all $v \in V$. We will prove, by induction, that $P(k)$ is true for all $0 \leq k \leq n-1$.

For the base case, we have $A_{0,s} = 0 = L_{0,s}$ and $A_{0,v} = \infty = L_{0,v}$ for all $v \in V \setminus \{s\}$, so $P(0)$ is true. For the inductive step, assume that $P(k-1)$ is true for some $1 \leq k \leq n-1$. We need to show that $P(k)$ is true.

In fact, by the inductive hypothesis, we know that $A_{k-1,u} = L_{k-1,u}$ for all $u \in V$. Thus, for any $v \in V$, we can see that

$$A_{k,v} = \min \left\{ \begin{array}{c} A_{k-1,v} \\ \min\limits_{(w,v)\in E} \{A_{k-1,w} + \ell(w,v)\} \end{array} \right\} = \min \left\{ \begin{array}{c} L_{k-1,v} \\ \min\limits_{(w,v)\in E} \{L_{k-1,w} + \ell(w,v)\} \end{array} \right\} = L_{k,v}$$

Hence, $P(k)$ is true, completing the induction.

Since $P(n-1)$ is true, we learn that $A_{n-1,v} = L_{n-1,v}$ for all $v \in V$. Combining this with $L_{n-1,v} = \delta(s,v)$, we get $A_{n-1,v} = \delta(s,v)$ for all $v \in V$.

Finally, let $Q(k)$ denote the statement that $\ell(B_{k,v}) = A_{k,v}$ for all $v \in V$. We will prove, by induction, that $Q(k)$ is true for all $0 \leq k \leq n-1$.

For the base case, we know that $\ell(B_{0,s}) = 0 = A_{0,s}$ and $\ell(B_{0,v}) = \infty = A_{0,v}$ for all $v \in V \setminus \{s\}$, so $Q(0)$ is true. For the inductive step, assume that $Q(k-1)$ is true for some $1 \leq k \leq n-1$. We need to show that $Q(k)$ is true.

If $A_{k,v} = A_{k-1,v}$, then we have $B_{k,v} = B_{k-1,v}$. This tells us that $\ell(B_{k,v}) = \ell(B_{k-1,v}) = A_{k-1,v} = A_{k,v}$ (by the inductive hypothesis).

Otherwise, we find a vertex $w' \in V$ such that $(w',v) \in E$ and $A_{k,v} = A_{k-1,w'} + \ell(w',v)$, then we have $B_{k,v} = B_{k-1,w'} \cup (w',v)$. This tells us that $\ell(B_{k,v}) = \ell(B_{k-1,w'}) + \ell(w',v) = A_{k-1,w'} + \ell(w',v) = A_{k,v}$.

Consequently, we learn that $\ell(B_{k,v}) = A_{k,v}$ for all $v \in V$. Hence, $Q(k)$ is true, completing the induction.

Since $Q(n-1)$ is true, we learn that $\ell(B_{n-1,v}) = A_{n-1,v}$ for all $v \in V$. Combining this with $A_{n-1,v} = \delta(s,v)$, we learn that $\ell(B_{n-1,v}) = \delta(s,v)$, so $B_{n-1,v}$ is a shortest $s \to v$ path for all $v \in V$.

Therefore, we can conclude that at the conclusion of the Bellman-Ford Algorithm, we have $A_{n-1,v} = \delta(s,v)$ and $B_{n-1,v}$ is a shortest $s \to v$ path for all $v \in V$. □

Subsequently, let's analyze the asymptotic time complexity of the Bellman-Ford Algorithm.

Theorem 4.7 (Running Time of Bellman-Ford Algorithm)**.** *Let $G = (V,E)$ be a weighted, directed graph with $|V| = n$ and $|E| = m$. Then, the Bellman-Ford Algorithm runs in $O((m+n)n)$ time (Roughgarden, 2019).*

Proof. Similar to the Dijkstra's Algorithm, we do not need to explicitly store and update the whole array B in practice. In fact, we only need to store a predecessor of v on a shortest path from s with at most k edges, and then update this value in each

iterations. At the conclusion of the algorithm, these predecessors can be used to easily construct the entire shortest paths. In this way, the update time for each entry of B becomes $O(1)$.

For each vertex $v \in V$, let $d_{\text{in}}(v)$ denote the in-degree of v, i.e. the number of vertices $u \in V$ such that $(u, v) \in E$. We know that $\sum_{v \in V} d_{\text{in}}(v) = m$.

For each outer iteration with $0 \leq k \leq n-1$, we need to carry out an exhaustive search over $1 + d_{\text{in}}(v)$ values. Processing each value requires $O(1)$ work, so the amount of work involved in each outer iteration is

$$\sum_{v \in V} (1 + d_{\text{in}}(v)) = n + \sum_{v \in V} d_{\text{in}}(v) = n + m$$

There are n outer iterations for $0 \leq k \leq n-1$, so the entire algorithm runs in $O((m+n)n)$ time. □

Remark 4.8. *If the input graph G is weakly connected, then the running time becomes $O((m+n)n) = O(mn)$, because $m \geq n-1$.*

Remark 4.9. *A slight improvement to the Bellman-Ford Algorithm is the use of early stopping. Specifically, if at some point, the values stop changing, i.e. $A_{k+1,v} = A_{k,v}$ for all $v \in V$, then we can ignore values higher than k and terminate the algorithm.*

The Bellman-Ford Algorithm fails to return desirable outputs when negative cycles appear in the input graph. However, we can extend the Bellman-Ford Algorithm to detect the existence of negative cycles.

Algorithm 4.10 (Extended Bellman-Ford Algorithm for Detecting Negative Cycles)**.** Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$, weight function $\ell : E \to \mathbb{R}$, and a source vertex s. The algorithm proceeds as follows.

- Run the Bellman-Ford Algorithm to obtain $A_{n-1,v}$ for all $v \in V$.
- For every $(u, v) \in E$, check if $A_{n-1,v} > A_{n-1,u} + \ell(u, v)$. If there is such an edge, return True. Otherwise, return False.

(Cormen et al., 2009)

Remark 4.11. *The extra step only involves $O(m)$ work. Consequently, this extended version has the same time complexity as the original Bellman-Ford Algorithm, which is $O((m+n)n)$.*

The following theorem will prove that the extended Bellman-Ford Algorithm correctly identifies the presence of negative cycles.

Theorem 4.12 (Correctness of the Extended Bellman-Ford Algorithm for Detecting Negative Cycles)**.** *Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$, weight function $\ell : E \to \mathbb{R}$, and a source vertex s. Then, Algorithm 4.10 returns True if and only if there exists a negative cycle in G (Cormen et al., 2009).*

Proof. First, suppose that G has no negative cycles. In this case, Theorem 4.6 implies that $A_{n-1,v} = \delta(s, v)$ for all $v \in V$.

Consider any edge $(u, v) \in E$. By following a shortest $s \to u$ and then moving along the edge (u, v), we can obtain a $s \to v$ path. By the definition of δ, we learn that $\delta(s, v) \leq \delta(s, u) + \ell(u, v)$. Thus, we have $A_{n-1,v} \leq A_{n-1,u} + \ell(u, v)$ for any edge $(u, v) \in E$, so the algorithm returns False.

Next, suppose that there exists a negative cycle in G. Assume for the sake of contradiction that the algorithm returns False, which implies that $A_{n-1,v} \leq A_{n-1,u} + \ell(u, v)$ for any edge $(u, v) \in E$.

Let $C = (v_0, v_1, \ldots, v_k, v_0)$ be a negative cycle in G, which means $\ell(C) < 0$.

For $0 \leq i \leq k - 1$, we have $(v_i, v_{i+1}) \in E$, which tells us that $A_{n-1,v_{i+1}} \leq A_{n-1,v_i} + \ell(v_i, v_{i+1})$. Hence, we learn $\ell(v_i, v_{i+1}) \geq A_{n-1,v_{i+1}} - A_{n-1,v_i}$ for $0 \leq i \leq k-1$. In addition, since $(v_k, v_0) \in E$, we have $A_{n-1,v_0} \leq A_{n-1,v_k} + \ell(v_k, v_0)$, which means $\ell(v_k, v_0) \geq A_{n-1,v_0} - A_{n-1,v_k}$. Now, we can see that

$$\ell(C) = \ell(v_k, v_0) + \sum_{i=0}^{k-1} \ell(v_i, v_{i+1}) \geq (A_{n-1,v_0} - A_{n-1,v_k}) + \sum_{i=0}^{k-1} (A_{n-1,v_{i+1}} - A_{n-1,v_i}) = 0$$

This contradicts the fact that $\ell(C) < 0$. We have reached a contradiction, so our assumption must have been wrong. Therefore, the algorithm returns True in this case.

Thus, Algorithm 4.10 returns True if and only if there exists a negative cycle in G. □

5 Floyd-Warshall Algorithm

The All-Pairs Shortest Path problem involves computation of shortest paths for every pair of vertices. A natural approach is applying the Bellman-Ford Algorithm to all n source vertices in the input graph. However, the time complexity of this approach is $O((m + n)n^2)$, which is highly problematic for large dense graphs. This section will present a beautiful solution to this challenge.

First, we also need the assumption that there are no negative cycles in the input graph. Then, we will utilize the proposition below to limit the search space for shortest paths to only cycle-free paths.

Proposition 5.1. *Let $G = (V, E)$ be a weighted, directed graph with no negative cycles. Then, if there is a path from u to v, then there is a shortest $u \to v$ path that contains no cycles.*

Proof. Let P denote a shortest $u \to v$ path with the smallest number of edges. We have $P = (v_0, v_1, \ldots, v_k)$ where $v_0 = u$ and $v_k = v$.

Assume for the sake of contradictions that P contains a cycle. Hence, there must be repeated vertices along P, which means $v_i = v_j$ for some $0 \leq i < j \leq k$.

Since G has no negative cycles, the cycle from v_i to v_j must have non-negative weight. By removing that cycle, we obtain a new $u \to v$ path $P' = (v_0, v_1, \ldots, v_{i-1}, v_i = v_j, v_{j+1}, \ldots, v_k)$ that has $\ell(P') \leq \ell(P)$. However, since P is a shortest $u \to v$ path, we know that $\ell(P) \leq \ell(P')$. This tells us that $\ell(P') = \ell(P)$, so P' is a shortest $u \to v$ path with fewer edges than P, which is a contradiction.

We have reach a contradiction, so our assumption must have been wrong. Hence, P contains at most $n - 1$ edges. Therefore, if there is a path from u to v, then there is a shortest $u \to v$ path that contains no cycles. □

Floyd-Warshall Algorithm is a famous algorithmic approach for solving the All-Pairs Shortest Path problem. Similar to the Bellman-Ford Algorithm, the Floyd-Warshall Algorithm also leverages the concept of dynamic programming, as shown below.

Algorithm 5.2 (Floyd-Warshall Algorithm)**.** Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$ and weight function $\ell : E \to \mathbb{R}$. Label the vertices as $V = \{1, 2, \ldots, n\}$. Let A be a $(n + 1) \times n \times n$ array such that $A_{k,u,v}$ will represent the minimum weight of a cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, k\}$ for each $0 \leq k \leq n$ and $u, v \in V$. In addition, let B be a $(n+1) \times n \times n$ array such that $B_{k,u,v}$ will represent a minimum-weight cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, k\}$ for each $0 \leq k \leq n$ and $u, v \in V$. The Floyd-Warshall Algorithm proceeds as follows.

- Initialize $A_{0,u,u} = 0$ and $B_{0,u,u} = (u)$ for all $u \in V$.
- Initialize $A_{0,u,v} = \ell(u, v)$ and $B_{0,u,v} = (u, v)$ for all $u, v \in V$ such that $(u, v) \in E$.
- Initialize $A_{0,u,v} = \infty$ and $B_{0,u,v} = ()$ for all $u, v \in V$ such that $(u, v) \notin E$.
- For $1 \leq k \leq n$ and $u, v \in V$, do the following steps:
  - Run the updates

$$A_{k,u,v} = \min \left\{ \begin{array}{c} A_{k-1,u,v} \\ A_{k-1,u,k} + A_{k-1,k,v} \end{array} \right\}$$

- If $A_{k,u,v} = A_{k-1,u,v}$, then set $B_{k,u,v} = B_{k-1,u,v}$. Otherwise, we have $A_{k,u,v} = A_{k-1,u,k} + A_{k-1,k,v}$, then set $B_{k,u,v} = B_{k-1,u,k} \cup B_{k-1,k,v}$.

- Return $A_{n,u,v}$ and $B_{n,u,v}$ for all $u, v \in V$.

(Roughgarden, 2019; Cormen et al., 2009)

To prove the correctness of the Floyd-Warshall Algorithm, we will analyze its optimal substructure.

Lemma 5.3 (Floyd-Warshall Optimal Substructure)**.** *Let $G = (V, E)$ be a weighted, directed graph with $V = \{1, 2, \ldots, n\}$. For some $1 \leq k \leq n$ and $u, v \in V$, let P denote a minimum-weight cycle-free $u \to v$ path in G with all internal vertices in $\{1, 2, \ldots, k\}$. Then, one of the following cases must occur.*

- *P is a minimum-weight cycle-free $u \to v$ path in G with all internal vertices in $\{1, 2, \ldots, k-1\}$.*

- *P is the concatenation of a minimum-weight cycle-free $u \to k$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$ and a minimum-weight cycle-free $k \to v$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$.*

(Roughgarden, 2019)

Proof. There are two cases.

- Case 1: k is not an internal vertex of P.

  In this case, P is a cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$. Let Q be a minimum-weight cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$. Since all interal vertices of Q are also in $\{1, 2, \ldots, k\}$, we have $\ell(Q) \geq \ell(P)$ by applying the definition of P. However, because all internal vertices of P are in $\{1, 2, \ldots, k-1\}$, we have $\ell(P) \geq \ell(Q)$ by applying the definition of Q. Thus, we learn that $\ell(P) = \ell(Q)$, so P is a minimum-weight cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$.

- Case 2: k is an internal vertex of P.

  Let P_1 and P_2 denote the $u \to k$ and $k \to v$ components of P, which means $\ell(P) = \ell(P_1) + \ell(P_2)$. Since P contains no cycles, k appears in P exactly once, so all the internal vertices of P_1 and P_2 are in $\{1, 2, \ldots, k-1\}$.

  In this case, we will show that P_1 is a minimum-weight cycle-free $u \to k$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$. In fact, assume that there is a shorter cycle-free $u \to k$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$, denoted

by P_1'. Then, the combined path $P' = P_1' \cup P_2$ has weight $\ell(P') = \ell(P_1') + \ell(P_2) < \ell(P_1) + \ell(P_2) = \ell(P)$. Hence, P' is a cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, k\}$ that is shorter than P, which is a contradiction. Thus, P_1 is a minimum-weight cycle-free $u \to k$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$. Using a similar proof, we learn that P_2 is a minimum-weight cycle-free $k \to v$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$.

Therefore, P is either a minimum-weight cycle-free $u \to v$ path in G with all internal vertices in $\{1, 2, \ldots, k-1\}$, or the concatenation of a minimum-weight cycle-free $u \to k$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$ and a minimum-weight cycle-free $k \to v$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$. □

This lemma results in a nice recurrence relation to compute shortest path weights.

Corollary 5.3.1 (Floyd-Warshall Recurrence)**.** *Let $G = (V, E)$ be a weighted, directed graph with $V = \{1, 2, \ldots, n\}$. For each $0 \leq k \leq n$ and $u, v \in V$, let $L_{k,u,v}$ denote the minimum weight of a cycle-free $u \to v$ path in G with all internal vertices in $\{1, 2, \ldots, k\}$ ($L_{k,u,v} = \infty$ if there are no such paths). Then, for every $1 \leq k \leq n$ and $u, v \in V$,*

$$L_{k,u,v} = \min \left\{ \begin{array}{c} L_{k-1,u,v} \\ L_{k-1,u,k} + L_{k-1,k,v} \end{array} \right\}$$

(Roughgarden, 2019)

Proof. Let P denote a minimum-weight cycle-free $u \to v$ path in G with all internal vertices in $\{1, 2, \ldots, k\}$. This means $\ell(P) = L_{k,u,v}$. Lemma 5.3 imples the following two cases.

- Case 1: P is a minimum-weight cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$.

  In this case, we have $\ell(P) = L_{k-1,u,v}$, so we get $L_{k,u,v} = L_{k-1,u,v}$.

- Case 2: P is the concatenation of P_1 and P_2, where P_1 is a minimum-weight cycle-free $u \to k$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$, and P_2 a minimum-weight cycle-free $k \to v$ path with all internal vertices in $\{1, 2, \ldots, k-1\}$.

  In this case, we have $\ell(P) = \ell(P_1) + \ell(P_2) = L_{k-1,u,k} + L_{k-1,k,v}$, so we get $L_{k,u,v} = L_{k-1,u,k} + L_{k-1,k,v}$.

Therefore, we can conclude that for every $1 \leq k \leq n$ and $u, v \in V$,

$$L_{k,u,v} = \min \left\{ \begin{array}{c} L_{k-1,u,v} \\ L_{k-1,u,k} + L_{k-1,k,v} \end{array} \right\}$$

□

Finally, the next theorem will provide a rigorous proof for the correctness of the Floyd-Warshall Algorithm.

Theorem 5.4 (Correctness of Floyd-Warshall Algorithm)**.** *Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$ and weight function $\ell : E \to \mathbb{R}$. Suppose that there are no negative cycles in G. Then, at the conclusion of the Floyd-Warshall Algorithm, we have $A_{n,u,v} = \delta(u, v)$ and $B_{n,u,v}$ is a shortest $u \to v$ path for all $u, v \in V$. (Roughgarden, 2019)*

Proof. The vertices of G are labelled as $V = \{1, 2, \ldots, n\}$. First, for each $0 \leq k \leq n$ and $u, v \in V$, let $L_{k,u,v}$ denote the minimum weight of a cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, k\}$. By Corollary 5.3.1, we learn that

$$L_{k,u,v} = \min \left\{ \begin{array}{c} L_{k-1,u,v} \\ L_{k-1,u,k} + L_{k-1,k,v} \end{array} \right\}$$

We can show that $L_{n,u,v} = \delta(u, v)$ for all $u, v \in V$. In fact, if v is not reachable from u, then $L_{n,u,v} = \delta(u, v) = \infty$. Otherwise, since there are no negative cycles in G, we can find a shortest $u \to v$ path that contains no cycles (By Proposition 5.1). Let P denote such a path.

Because $L_{n,u,v}$ is the minimum weight of a cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, n\}$, we have $L_{n,u,v} \leq \ell(P)$. However, since P is a shortest $u \to v$ path, we know that $L_{n,u,v} \geq \ell(P)$. Thus, we learn that $L_{n,u,v} = \ell(P) = \delta(u, v)$. Consequently, we can see that $L_{n,u,v} = \delta(u, v)$ for all $u, v \in V$.

Next, let $P(k)$ denote the statement that $A_{k,u,v} = L_{k,u,v}$ for all $u, v \in V$. We will prove, by induction, that $P(k)$ is true for all $0 \leq k \leq n$.

For the base case, we have $A_{0,u,u} = 0 = L_{0,u,u}$ for all $u \in V$, $A(0, u, v) = \ell(u, v) = L_{0,u,v}$ for all $u, v \in V$ where $(u, v) \in E$, and $A(0, u, v) = \infty = L_{0,u,v}$ for all $u, v \in V$ where $(u, v) \notin E$. Hence, $P(0)$ is true. For the inductive step, assume that $P(k - 1)$ is true for some $1 \leq k \leq n$. We need to show that $P(k)$ is true.

In fact, by the inductive hypothesis, we know that $A_{k-1,u,v} = L_{k-1,u,v}$ for all $u, v \in V$. Thus, for any $u, v \in V$, we can see that

$$A_{k,u,v} = \min \left\{ \begin{array}{c} A_{k-1,u,v} \\ A_{k-1,u,k} + A_{k-1,k,v} \end{array} \right\} = \min \left\{ \begin{array}{c} L_{k-1,u,v} \\ L_{k-1,u,k} + L_{k-1,k,v} \end{array} \right\} = L_{k,u,v}$$

Hence, $P(k)$ is true, completing the induction.

Since $P(n)$ is true, we learn that $A_{n,u,v} = L_{n,u,v}$ for all $u, v \in V$. Combining this with $L_{n,u,v} = \delta(u, v)$, we get $A_{n,u,v} = \delta(u, v)$ for all $u, v \in V$.

Finally, let $Q(k)$ denote the statement that $\ell(B_{k,u,v}) = A_{k,u,v}$ for all $u, v \in V$. We will prove, by induction, that $Q(k)$ is true for all $0 \leq k \leq n$.

For the base case, we have $\ell(B_{0,u,u}) = 0 = A_{0,u,u}$, $\ell(B_{0,u,v}) = \ell(u, v) = A_{0,u,v}$ for all $u, v \in V$ where $(u, v) \in E$, and $\ell(B_{0,u,v}) = \infty = A_{0,u,v}$ for all $u, v \in V$ where $(u, v) \notin E$. Hence, $Q(0)$ is true. For the inductive step, assume that $Q(k-1)$ is true for some $1 \leq k \leq n$. We need to show that $Q(k)$ is true.

If $A_{k,u,v} = A_{k-1,u,v}$, then we have $B_{k,u,v} = B_{k-1,u,v}$. This tells us that $\ell(B_{k,u,v}) = \ell(B_{k-1,u,v}) = A_{k-1,u,v} = A_{k,u,v}$ (by the inductive hypothesis).

Otherwise, when $A_{k,u,v} = A_{k-1,u,k} + A_{k-1,k,v}$, we have $B_{k,u,v} = B_{k-1,u,k} \cup B_{k-1,k,v}$. This tells us that $\ell(B_{k,u,v}) = \ell(B_{k-1,u,k}) + \ell(B_{k-1,k,v}) = A_{k-1,u,k} + A_{k-1,k,v} = A_{k,u,v}$ (by the inductive hypothesis).

Consequently, we learn that $\ell(B_{k,u,v}) = A_{k,u,v}$ for all $u, v \in V$. Hence, $Q(k)$ is true, completing the induction.

Since $Q(n)$ is true, we learn that $\ell(B_{n,u,v}) = A_{n,u,v}$ for all $u, v \in V$. Combining this with $A_{n,u,v} = \delta(u, v)$, we learn that $\ell(B_{n,u,v}) = \delta(u, v)$, so $B_{n,u,v}$ is a shortest $u \to v$ path for all $u, v \in V$.

Therefore, we can conclude that at the conclusion of the Floyd-Warshall Algorithm, we have $A_{n,u,v} = \delta(u, v)$ and $B_{n,u,v}$ is a shortest $u \to v$ path for all $u, v \in V$. □

After that, let's calculate the time complexity of the Floyd-Warshall Algorithm.

Theorem 5.5 (Running Time of Floyd-Warshall Algorithm)**.** *Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$. Then, the Floyd-Warshall Algorithm runs in $O(n^3)$ time (Roughgarden, 2019).*

Proof. Similar to the previously mentioned algorithms, we do not need to explicitly store and update the whole array B in practice. In fact, we only need to store a predecessor of v on a minimum-weight a cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, k\}$, and then update this value in each iterations. At the conclusion of the algorithm, these predecessors can be used to easily construct the entire shortest paths. In this way, the update time for each entry of B becomes $O(1)$.

The algorithm updates two $(n+1) \times n \times n$ arrays. Computing each entry involves $O(1)$ work. Outside the nested loops, the initialization steps required $O(n^2)$ work. Thus, the entire algorithm runs in $O(2(n+1)n^2 + n^2) = O(n^3)$ time. □

Moreover, similar to the Bellman-Ford Algorithm, the Floyd-Warshall Algorithm can also be extend to detect the existence of negative cycles.

Algorithm 5.6 (Extended Floyd-Warshall Algorithm for Detecting Negative Cycles)**.** Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$ and weight function $\ell : E \to \mathbb{R}$. The algorithm proceeds as follows.

- Run the Floyd-Warshall Algorithm to obtain $A_{n,u,v}$ for all $u, v \in V$.
- For every $u \in V$, check if $A_{n,u,u} < 0$. If there is such a vertex, return True. Otherwise, return False.

(Roughgarden, 2019)

Remark 5.7. *The extra step only involves $O(n)$ work. Consequently, this extended version shares the same asymptotic time complexity with the original Floyd-Warshall Algorithm, which is $O(n^3)$.*

We will prove that the extended Floyd-Warshall Algorithm correctly identifies the presence of negative cycles. To complete this process, we need two lemmas.

Lemma 5.8. *Let $G = (V, E)$ be a weighted, directed graph with weight function $\ell : E \to \mathbb{R}$. Suppose that there exists a negative cycle in G. Then, G has a negative cycle with no repeated vertices other than its endpoints (Roughgarden, 2019).*

Proof. Let C denote a negative cycle with the smallest number of edges. We have $C = (v_0, v_1, \ldots, v_k, v_0)$. Assume for the sake of contradictions that C contains some repeated vertices other than its endpoints, $v_i = v_j$ for some $0 \leq i < j \leq k$.

Let $C_1 = (v_i, v_{i+1}, \ldots, v_{j-1}, v_j)$ and $C_2 = (v_j, v_{j+1}, \ldots, v_k, v_0, \ldots, v_{i-1}, v_i)$. Then, C is the concatenation of the two cycles C_1 and C_2. This tells us that $\ell(C_1) + \ell(C_2) = \ell(C) < 0$. Thus, there exists $i \in \{1, 2\}$ such that $\ell(C_i) < 0$. However, C_i is a negative cycle with fewer edges than C, which is a contradiction.

We have reach a contradiction, so our assumption must have been wrong. Hence, C contains no repeated vertices other than its endpoints. G has a negative cycle with no repeated vertices other than its endpoints. □

Lemma 5.9. *Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$ and weight function $\ell : E \to \mathbb{R}$. Suppose we run the Floyd-Warshall Algorithm to obtain $A_{n,u,v}$ for all $u, v \in V$. Then, for any $0 \leq k \leq n$ and $u, v \in V$, $A_{k,u,v}$ is equal to the minimum weight of a cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, k\}$.*

Proof. The vertices of G are labelled as $V = \{1, 2, \ldots, n\}$. First, for each $0 \leq k \leq n$ and $u, v \in V$, let $L_{k,u,v}$ denote the minimum weight of a cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, k\}$.

The first half of the proof of Theorem 5.4 already showed that $A_{k,u,v} = L_{k,u,v}$ for all $0 \leq k \leq n$ and $u, v \in V$ (by induction). Therefore, by following that proof, we can see that for any $0 \leq k \leq n$ and $u, v \in V$, $A_{k,u,v}$ is equal to the minimum weight of a cycle-free $u \to v$ path with all internal vertices in $\{1, 2, \ldots, k\}$. □

Theorem 5.10 (Correctness of the Extended Floyd-Warshall Algorithm for Detecting Negative Cycles)**.** *Let $G = (V, E)$ be a weighted, directed graph with $|V| = n$ and weight function $\ell : E \to \mathbb{R}$. Then, Algorithm 5.6 returns True if and only if there exists a negative cycle in G (Roughgarden, 2019).*

Proof. The vertices of G are labelled as $V = \{1, 2, \ldots, n\}$.

First, suppose that G has no negative cycles. In this case, Theorem 5.4 implies that $A_{n,u,v} = \delta(u, v)$ for all $u, v \in V$. Thus, we have $A_{n,u,u} = \delta(u, u) = 0$ for all $u \in V$, so the algorithm return False in this case.

Next, suppose that there exists a negative cycle in G. Then, we can find a negative cycle with no repeated vertices other than its endpoints (by Lemma 5.8). Let C denote such a cycle.

Pick a vertex k in C with the largest label, and pick another vertex $v \neq k$ in C. Let P_1 and P_2 denote the $k \to v$ and $v \to k$ components of C respectively. Because k is the largest vertex in C, and C contains no repeated vertices other than its endpoints, we learn that P_1 and P_2 are cycle-free $k \to v$ and $v \to k$ paths with all internal vertices in $\{1, 2, \ldots, k-1\}$. By applying Lemma 5.9, we obtain $\ell(P_1) \geq A_{k-1,k,v}$ and $\ell(P_2) \geq A_{k-1,v,k}$.

In addition, we know that

$$A_{k,v,v} = \min \left\{ \begin{array}{c} A_{k-1,v,v} \\ A_{k-1,v,k} + A_{k-1,k,v} \end{array} \right\}$$

This tells us that $A_{k,v,v} \leq A_{k-1,v,k} + A_{k-1,k,v} \leq \ell(P_2) + \ell(P_1) = \ell(C) < 0$. Therefore, the algorithm return True in this case.

Thus, Algorithm 5.6 returns True if and only if there exists a negative cycle in G. □

6 Conclusion and Further Research

This paper has presented a rigorous analysis of three foundational algorithms for solving the shortest-path problem in weighted, directed graphs. Together, Dijkstra's, Bellman-Ford, and Floyd-Warshall's algorithms form a powerful toolkit, each tailored to different constraints and problem scopes.

- Dijkstra's algorithm stands out for its efficiency in the common scenario of non-negative edge weights, making it a cornerstone of applications like modern route-planning systems.
- The Bellman-Ford algorithm provides critical flexibility by handling graphs with negative edge weights, and crucially, it can detect the presence of negative cycles—a feature essential for applications in areas like financial arbitrage analysis.
- Finally, the Floyd-Warshall algorithm offers a powerful solution to the all-pairs shortest path problem, delivering a complete distance matrix that is invaluable for network analysis, albeit at a higher computational cost.

By formally proving the correctness and analyzing the time complexity of each, this work solidifies the theoretical underpinnings that make these algorithms indispensable tools in computer science, operations research, and network engineering.

The study of shortest-path algorithms remains a vibrant and evolving field. While the algorithms discussed here are foundational, they serve as the basis for numerous advanced techniques and ongoing research. For instance, Moore (1959) and Yen (1970) utilize some clever orderings of vertex and edge candidates to reduce the number of iterations in the Bellman-Ford Algorithm. In addition, for graphs that are not very dense, the Johnson's Algorithm modifies the All-Pairs Shortest Path problem in a clever way to apply one execution of the Bellman-Ford Algorithm, followed by $n-1$ executions of the Dijkstra's Algorithm. Its total runtime is $O(mn \log n)$ (Roughgarden, 2019). At the same time, Pettie (2004) develops a hierarchy-based approach to calculate all-pairs approximate shortest paths and then refine them to obtain exact ones, and this method runs in $O(mn + n^2 \log \log n)$ time. Another solution is the A^* search algorithm, which leverages heuristic functions for efficient graph traversal and path search (Norvig and Russell, 2011). Furthermore, in practice, we often harness parallelism and concurrency to speed up the running time of the above shortest path finding algorithms.

Many recent research projects have focused on developing randomized algorithms, a class of algorithms that produce correct outputs with high probability. For example, Seidel (1992) offers such a solution with the help of some randomized matrix multiplication techniques. Likewise, Saerens et al. (2009) builds two randomized models based on Markov chains to search for shortest paths. Another promising algorithm design

paradigm is leveraging spectral graph theory for computing shortest paths. An example is the Steinerberger's Algorithm, which takes advantage of the eigenvectors and eigenvalues of the Laplacian matrix to estimate shortest paths (Steinerberger, 2021). Meawhile, Peng (2013) introduces an alternative approach that reduces the shortest path problem to the minimization of several Lasso objectives.

References

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. 2009. *Introduction to Algorithms*. Computer science. McGraw-Hill.

Edward F Moore. 1959. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959*, pages 285–292.

P. Norvig and S. Russell. 2011. *Artificial Intelligence: A Modern Approach*. Pearson Education.

Richard Peng. 2013. *Algorithm design using spectral graph theory*. Ph.D. thesis, Carnegie Mellon University.

Seth Pettie. 2004. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312(1):47–74.

T. Roughgarden. 2018. *Algorithms Illuminated: Graph algorithms and data structures. Part 2*. Algorithms Illuminated. Soundlikeyourself Publishing LLC.

T. Roughgarden. 2019. *Algorithms Illuminated: Greedy algorithms and dynamic programming. Part 3*. Algorithms Illuminated. Soundlikeyourself Publishing, LLC.

Marco Saerens, Youssef Achbany, Francois Fouss, and Luh Yen. 2009. Randomized shortest-path problems: Two related models. *Neural Computation*, 21(8):2363–2404.

Raimund Seidel. 1992. On the all-pairs-shortest-path problem. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 745–749.

Stefan Steinerberger. 2021. A spectral approach to the shortest path problem. *Linear Algebra and its Applications*, 620:182–200.

Jin Y Yen. 1970. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics*, 27(4):526–530.